VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvovanie individuálnej odbornej praxe

Individual Professional Practice in the Company

2018 Lukáš Mrvečka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Mrvečka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: KROS a.s.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Janoušek**

Konzultant bakalářské práce: Ing. Martin Luhový

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 19. apríla 2018

.....

Súhlasím so zverejnením tejto bakalárskej práce podľa požiadaviek čl. 26, odst. 9 Študijného a skúšobného poriadku pre štúdium v bakalárskych programoch VŠB-TU Ostrava.

V Žiline 13. apríla 2018 .....

Týmto by som chcel poďakovať kolegom z práce, ale hlavne odbornému konzultantovi Ing. Martinovi Luhovému, za ich odbornú pomoc pri vytváraní tejto práce. Moja vďaka patrí aj Ing. Janovi Janouškovi za jeho ústretový prístup a pomoc.

Abstrakt

Táto práca popisuje odbornú prax vo firme Kros a.s. V rámci praxe som pracoval na rôznych zaujímavých projektoch. Moja práca je zameraná na využitie OCR a dodatočné spracovanie výstupu do podoby vhodnej na spracovanie v účtovnom programe a na navrhnutie nástroja na vytváranie šablón. Existuje veľké množstvo knižníc, ktoré poskytujú možnosti OCR, ja som si vybral Tesseract pretože je to OpenSource knižnica pôvodne vyvíjaná spoločnosťou HP, neskôr prevzatá spoločnosťou Google a má pomerne dobrú úspešnpsť v rozpoznávaní znakov. V úvode práce sa venujem popisu firmy, v ktorej som prax vykonával a v krátkosti aj jej produktom. V ďalšej časti popisujem zadanie a aj samotné vypracovanie spolu s riešením problémov, ktoré vznikli počas práce. Na záver som zhrnul získané vedomosti pri práci a využité poznatky, nadobudnuté počas štúdia.

Kľúčové slová: C#, .Net, Tesseract, rozpoznávanie textu, účtovnícky software

Abstract

This bechelor thesis describes my individual practise in company Kros a.s. I was working on various interesting projects during practise. My work is focused on using OCR and extra processing of the output to the form suitable for economic software and create template creation tool. There are large number of libraries that provide OCR capabilities, I have chosen Tesseract because it is OpenSource library from Google and has a fairly good success in character recognition. At the beggining I introduce company where i was working and in short it's products. In the next part I explain task and solutions with problems which occured during implementation. At the end I explain sumary of what I have to learn to achieve goal and what I already know from school.

Key Words: C#, .Net, Tesseract, text recognition, economic software

Obsah

Zoznam použitých skratiek a symbolov

PDF	–	Portable document format
SQL	–	Structured Query Language
ISRI	–	Information Science Research Institute
UNLV	–	University of NEvada, Las Vegas
ERP	–	Elektronic registration repository
FIT	–	Fond of Inovation and Technologies
BSD	–	Berkeley Software Distribution
OS	–	Operating system
UI	–	User Interface (Uzívateľské rozhranie)
PLC	–	AIMP Playlist File
PXL	–	Pocket Excel spreadsheet
XPS	–	XML Paper Specification

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

1 Úvod

Odbornú prax som vykonával vo firme Kros a.s. V rámci praxe som pracoval na rôznych inšpiratívnych projektoch. Moja práca je zameraná na využitie OCR a dodatočné spracovanie výstupu do podoby vhodnej na spracovanie v účtovnom programe a na navrhnutie nástroja na vytváranie šablón. Veľa účtovníkov, účtovných firiem denne dostáva a odosiela doklady, ktoré musí niekto na druhej strane spracovať. Použitím takéhoto softvéru by nutnosť ručného spracovávania dokladov odpadla. Existuje veľké množstvo knižníc, poskytujúcich možnosti OCR, avšak mňa konkrétne zaujal Tesseract, pretože ide o OpenSource knižnicu pôvodne vyvíjanú spoločnosťou HP, neskôr prevzatú spoločnosťou Google s pomerne dobrou úspešnosťou v rozpoznávaní znakov. V súčastnosti je veľa softvérov, ktoré dokážu použiť nejaké OCR a zobraziť výstup ako neštruktúrované dáta. Práve to je problém, ktorý chceme eliminovať. Tento problém by malo vyriešiť dodatočné spracovanie textu, v ktorom podľa kľúčových slov hľadáme k nim hodnoty. Toto riešenie by bolo dostačujúce ak by účtovné doklady mali svoj "suit code". Tu sa dostávame k ďalšej prekážke a tou je množstvo spracovávaných dát a široké spektrum štýlov dokladov. Vo firme som bol súčasťou scrum tímu a spolupracovali sme na rôznych zaujímavých projektoch. Mal som možnosť sa naučiť nové veci, ale aj zúžitkovať vedomosti nadubudnuté počas školy. Odbornú prax vo firme som si vybral hlavne z dôvodu, že som vo firme už nejaký čas pracoval a mal som možnosť spojiť prácu zo školou.

2 Firma

História firmy Kros sa začala písať už v 90-tých rokoch minulého storočia keď sa čerství absolventi Žilinskej univerzity rozhodli naštartovať svoje podnikateľské plány. Spolu s kamarátmi začali vyvíjať program na riadenie stavebnej výroby vo firme ODIS. Neskôr v roku 1995 sa rozhodli založiť firmu Kros, ktorá je pomenovaná po hlavnom stavebnom kameni firmy, po programe CENKROS. V súčasnosti má viac ako 200 zamestnancov, 90 tisíc zákazníkov po celom Slovensku a veľký apetít na ďalší rast.

2.1 Produkty firmy

Firma Kros ponúka hneď niekoľko produktov hlavne v oblasti ekonomiky a účtovníctva. Spomeniem len pár z nich, ostatné si môžete pozrieť na oficiálnych stránkach.[1]
Hlavným ťahúňom a zároveň najväčším projektom je program OMEGA - podvojné účtovníctvo. Multifunkčný nástroj na spracovanie účtovníctva, komplexnú skladovú evidenciu a spracovávanie fakturačných dokladov ich zaúčtovanie, úhradu, tlač a odosielanie klientom.
Ďalej by som rád spomenul program na oceňovanie a riadenie stavebnej výroby - CENKROS 4. Najpredávanejší stavebný softvér na Slovensku. Zastrešuje všetky činnosti spojené s prípravou a realizáciou stavebnej zákazky. Program je určený na vytváranie cenových ponúk s aktuálnymi cenami materiálov a stavebných prác.
OLYMP - mzdy a personalistika - Program je určený na výpočet miezd vašich zamestnancov a vedenie personalistiky. V programe rýchlo a ľahko vypočítate mzdy zamestnancov a ich odmeny pre všetky typy pracovných pomerov.
ALFA plus - jednoduché účtovníctvo - moderný účtovnícky softvér pre každého, kto potrebuje viesť jednoduché účtovníctvo alebo zjednodušenú daňovú evidenciu.
iKROS - online faktúry - bezplatná online služba, ktorá umožňuje pohodlne vystavovať faktúry cez internet, posielať ich emailom alebo tlačiť a sledovať ich úhrady.

2.2 Pracovné zaradenie

V čase keď som sa hlásil o zamestnanie som nemal žiadnu prax a kolegom to bolo hneď jasné. Po prijatí mi bol pridelený mentor, ktorý so mnou spolupracoval na úlohách, postupne som ale dostával viac priestoru a z mentora sa stal kolega. Postupne som začal pracovať ako FrontEnd Developer programu OMEGA v .NET framworku.

[1] `http://www.kros.sk`

3 Popis projektu

Hlavným cieľom projektu je vytvoriť desktopovu aplikáciu, v ktorej užívateľ vyberie, ktoré dokumenty chce zaúčtovať a program to spraví za neho. Účtovníci a veľké účtovné firmy dostávajú denne množstvo došlých dokladov, ktoré musia ručne prepísať do programu. Výhodou programu je, že po nastavení a spustení pracuje na pozadí a nevyžaduje akýkoľvek zásah človeka. Program načíta, spracuje a po spracovaní daný doklad importuje do účtovníckeho softwéru. Rozpoznané dáta môžme skontrolovať poprípade, ak bude nutné upraviť ešte pred importom. Program je zameraný na doklady fakturácie. Aplikacia je schopná spracovávať rôzne typy obrázkov ako napríklad bmp, jpg, jpeg, png, gif, tiff a taktiež súbory formátu pdf. Po spracovaní má užívateľ možnosť skontrolovať rozpoznané dáta a uložiť ich do databázy ako vzor pre dalšie spracovávanie. Pred spracovaním súboru program podľa pozícii zisťuje či môže použiť nejaký vzor z databázy alebo bude vyhľadávať štýlom "BruteForce". Databáza nie je nutnosť, pre aplikaciu slúži len pre ukladanie vzorov. Ak nie je pripojená program vždy pôjde štýlom "BruteForce".

3.1 Ostatné projekty

Počas praxe som spolupracoval aj na projektoch, ktoré nie sú v tejto práci uvedené. Jednalo sa najmä o opravu chýb v programe (BugFixing), ale aj menšie nemenej zaujímavé projekty ako zbieranie štatistík využívania funkcii v programe , vytváranie tlačových zostáv pomocou DevExpress nástroja alebo plnenie individuálnych požiadavok zákazníkov na import/export údajov z databáz. Tieto úlohy mi zabrali asi 50% času a popri nich som vytváral zadanú úlohu.

4 Podobné riešenia

V dnešnom uponáhľanom svete sa snažíme šetriť čas doslova na každom kroku. Inak tomu nie je ani pri vývoji softvérov. V rámci best-practises sa programy vyvíjajú tak aby bol kód čistý a podľa možností efektívny. Programy sú čoraz komplexnejšie, šetria čas, ktorý by sme inak strávili vykonávaním úloh, ktoré sú dnes už plne automatizované. V podobnom duchu sa nesú aj niektoré ekonomické programy, ktoré značne pokročili v efektivite využitia nových technológii a tým prispeli k vývoju programov, ktoré zasa o niečo viac odbremenia ľudí. V prípade ekonomických softvérov je to hlavne ručné prepisovanie došlých dokladov do programu. Tento problém som riešil v rámci odbornej praxe vo firme Kros. Nie sme prví koho niečo podobné napadlo a chcel proces ručného účtovania zefektívniť alebo dokonca úplne odstrániť. Z dostupných zdrojov môžme povedať, že programy, ktoré majú túto možnosť už zapracovanú využívajú prevažne šablóny a dodatočnú kontrolu. Strojovo je takmer nemožné dosiahnuť 100% presnosť kvôli variabilite dokladov, kvalite a pod., a preto je po rozoznaní nutná ešte kontrola údajov a ich korekcia.

4.1 Archiles

Revolučná platforma, ktorá automatizuje spracovanie účtovných a firemných dokladov. Automatické rozpoznávanie a vyťažovanie účtovných a bankových údajov z dokladov (faktúry, bločky, dodacie listy), automatické účtovanie predpisov (predkontácií) a nahrávanie údajov do účtovného alebo ERP systému. Integrácia s účtovnými systémami: Stormware Pohoda, Kros Alfa, Kros Omega, Money S3, MRP K/S, MRP Vizuálny, Asseco Helios Orange, ISDOC, CAISE. Stačí nahrať doklady cez prehliadač, mobilnú aplikáciu alebo ich poslať na e-mail do systému Archiles a služba Vám vráti údaje do Vášho účtovníctva. Ak nemáte doklady v elektronickej podobe, stačí ich odfotiť mobilom alebo pohodlne zdigitalizovať prostredníctvom skenera s podávačom na papiere. Po stlačení tlačidla „SCAN“ sa doklady nahrajú do systému Archiles a následne do účtovníctva. Možnosť automatického účtovania dokladov je v prípade programu Archiles riešená rôznymi spôsobmi aby dosiahli čo najvyššiu presnosť. Po rozpoznaní sa dáta posielajú do verifikačného centra, kde personál porovná, skontroluje a opravy vyťažený text s originálom. Až po kontrole sú dáta odoslané do účtovného softvéru.[2]

4.2 Datamolino

Datamolino je služba, ktorá spracuje všetky prichádzajúce faktúry a bločky bez prepisovania. Zároveň ich zatriedi v rámci on-line archívu, kde sú stále na dosah. Datamolino šetrí čas i peniaze, eliminuje chyby spôsobené manuálnym prepisovaním údajov. Poskytuje tak výhody veľkých podnikových riešení aj pre malé a stredné podniky. Za Datamolinom, ktoré vyrástlo v bratislavskom co-workingovom centre The Spot, stojí tím približne dvadsiatich ľudí okolo lídrov Andreja

[2] `http://www.archiles.sk/`

Glézla, Jána Koreckého a Michala Pauloviča. Datamolino má vyškolených operátorov, ktorí dáta validujú. Účtovných pracovníkov firiem teda odbremenia nielen od prepisovania faktúr, ale aj od kontroly dát. Chybovosť je určite výrazne nižšia ako pri manuálnom prepisovaní faktúr. Okrem iného s Datamolinom sa kancelária stáva Paperless. [3] [4]

4.3 DOCKitIN

Podobnou cestou sa vydáva aj služba DOCKitIN. Prináša automatické prenesenie a zaúčtovanie zoskenovanej faktúry bez manuálneho prepisovania údajov do Profit365. Ponúka webovú a mobilnú aplikáciu úplne zadarmo. Po registrácii získate skušobný počet strán elektronického vyťažovania. Po minutí kreditu je nutné si ďalšie stránky dokúpiť. Znižuje chybovosť pri prepisovaní údajov, šetrí čas a miesto pri evidencii. Služba je priamo prepojená s ekonomickým softvérom Profit365, ale ponúka aj možnosť exportovať údaje do .csv súboru, ktorý je možné importovať aj do iných softvérov. Všetky dôležité údaje sú na jednom mieste, dostupne z mobilného telefónu, tabletu alebo webu. Rozpoznávanie dôležitých údajov je realizované pomocou šablón a po vyťažení sú dáta ešte skontrolované a opravené personálom.

[3] `https://www.etrend.sk/ekonomika/slovensky-start-up-s-polmilionom.html`
[4] `https://www.pcrevue.sk/a/Predstavujeme-slovenske-startupy--Datamolino-je-Dropbox-na-blocky-a-faktury`

5 OCR - Optical Character Recognition (Optické rozpoznávanie znakov)

OCR je metóda prekladu obrazu do textovej a editovateľnej podoby napr. do ASCII znakov. Príkladom môžu byť skeny a ich rozpoznanie a príprava na ďalšie spracovanie výstupného textu. Je to ako kombinácia ľudského oka a mysle. Zatiaľ čo okom môžme text vidieť, až mozok spracuje to, čo vidíme. Pri vývoji OCR systémov, ale aj pri ich samotnom použití, môžme naraziť na niekoľko problémov:

- Pri niektorých znakoch existuje len malý viditeľný rozdiel pre počítače. Napríklad môže byť tažké určiť, či sa jedná o písmeno O alebo číslicu 0.
- Aby systém dosiahol čo najlepšiu presnosť vstupné dáta by mali mať dobrý kontrast aby bolo jasné čo je text a čo nie (nie je vhodné aby sa text prekrýval alebo bol vytlačený na pozadí, ktoré pôsobý rušivo).

OCR technológie majú veľmi široké využitie. Používajú sa napr. na rozpoznávanie notového zápisu, špeciálnych znakov (bankovky), podpis atď. Dnes je na trhu k dispozíci mnoho OCR softvérov, ale len málo z nich je OpenSource. Využitie nachádzaju nielen na bežných počítačoch, ale aj na webe či serveroch. Presnosť dnešných systémov využívajúcich OCR sa pohybuje niekde medzi 70% až 98%.

5.1 OCR softvér

OCR softvér predstavuje samotný program určený na rozpoznávanie znakov v texte. Obsahuje prvky umelej inteligencie a tiež lingvistické prvky, ktoré slúžia na kontrolu správnosti preložených slov a viet tzv. MeanConfidence(miera spokojnosti s prekladom). Presnosť výstupu závisí od zložitosti softvéru a kvality daného obrázka. Dôležitým krokom pri použití OCR je nastavenie jazyka, v ktorom sa text bude rozpoznávať (slovenčina, angličtina, čestina...). Je to z toho dôvodu aby vedel správne vyhodnotiť špeciálne znaky pre daný jazyk napr. diakritika a aby použil správny jazykový slovník na výstupný text. Samostatnou kapitolou sú softvéry schopné rozpoznávať rukou písaný text, kde je treba dávať pozor na font, štýl písma, sklon písma atď. Väčšinou sú vybavené schopnosťou naučiť sa a prispôsobiť danému štýlu.
Známe OCR softvéry:

- GOCR
- OmiPage
- Microsoft OneNote
- Nicromsoft OCR

- Google VISION
- ABBYY FineReader OCR
- Nuance
- OpenText

5.1.1 Ako OCR funguje?

Pri spracovávaní OCR sa naskenovaný obrázok alebo bitová mapa analyzuje na svetlé a tmavé časti, aby sa idetifikovali všetky abecedné písmená alebo číslice. V prípade že program rozpozná nejaký znak konvertuje ho na ASCII kód. Na urýchlenie OCR sa používajú špeciálne čipy.

5.1.2 Predspracovanie

Pred samotným spracovaním je potrebné dokument dostatočne vyčistiť, aby sme získali lepší výstup. OCR softvéry používajú rôzne techniky na zlepšenie kvality snímkov napr. odstránenie šumu, vyrovnanie obrázkov, odstránenie akýchkoľvek rozostrení, zvýšenie kontrastu a zaostrenie textu, aby proces rozpoznania bol čo najpresnejší.

5.1.3 Prvý prechod

Väčšina moderných OCR systémov používa princíp dvoch prechodov (two-pass principle). To znamená, že softvér prechádza každý dokument dvakrát. Prvý prechod sa vykonáva bez predošlých znalostí dokumentu. Program skenuje bežne sa vyskytujúce symboly, vyberá tie, s ktorými si je na 99% istý a rozdeľuje ich podľa ich základných tvarov. V ostatných slovách sa zameriava na najzreteľnejšie symboly a začne budovať knižnicu, ktorá vyzerá ako rukopis konkrétneho dokumentu.

5.1.4 Druhý prechod

Úlohou druhého prechodu je zobrať znaky, pre ktoré má program vysokú dôveru a použiť interný slovník na odhad slov. Najlepšie OCR systémy sú schopné kontrolovať gramatiku a syntax rovnako ako dobrý textový editor. Vysoká presnosť nikdy nie je zárukou, avšak metóda dvoch prechodov može zaistiť prijateľnú presnosť a metodika sa neustále zlepšuje.

5.2 Analýza OCR

Existuje veľa spôsobov, ako kvantifikovať odchýlku medzi výstupom z OCR a správnym textom. Základným meradlom odchylky je úsilie, ktoré musí človek vynaložiť, aby upravil text do správnej podoby. Konkrétne je to počet úprav (vloženie, zmazanie alebo nahradenie znakov), ktoré je

nutné vykonať. Toto číslo môžme chápať aj ako počet chýb OCR systému. Presnosť vyjadrujeme podľa rovnice 1 [6]

$$(Z \ - E)/Z \tag{1}$$

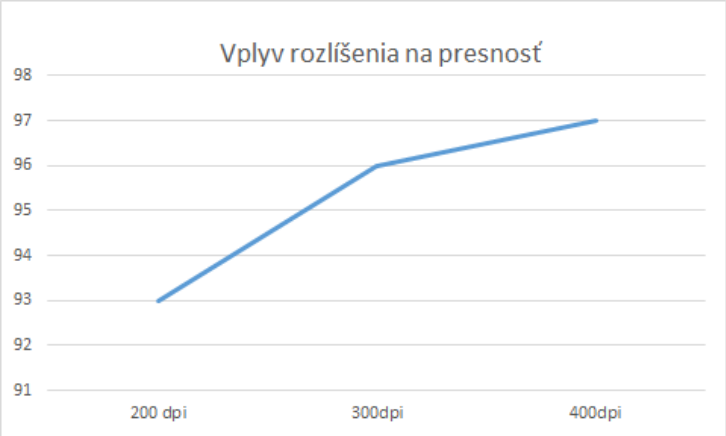
kde Z predstavuje počet znakov a E počet chýb

Pre väčšinu aplikácii nie je rýchlosť tak podstatná, ako presnosť, avšak využitie nájdu aj také aplikácie, ktoré vsadili práve na rýchlosť. V prípade, že presnosť systémov je takmer identická, rýchlosť sa stáva dôležitým faktorom. Rovnica č. 2 vyjadruje presnosť OCR. [6]

$$(Z \ - P * E)/T \tag{2}$$

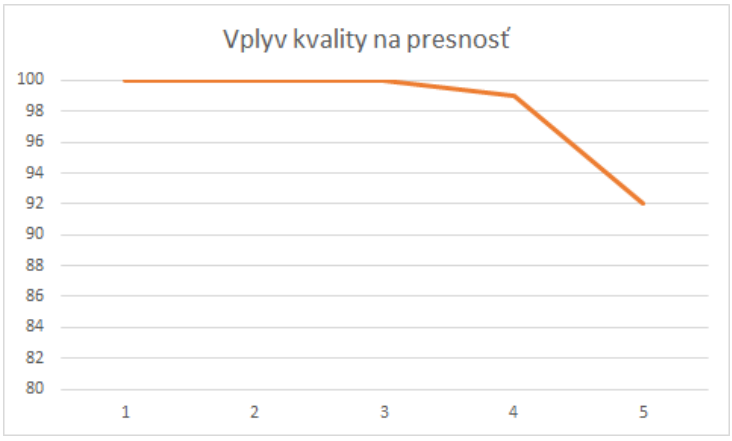
kde Z predstavuje počet znakov, E počet chýb, T čas v sekundách a P je trest za každú chybu

Kvalita snímok je dôležitým faktorom pri OCR systémoch. Na grafe 1 môžeme vidieť vplyv hĺbky snímky na presnosť OCR.



Obr. 1: Všeobecná závislosť presnosti na rozlíšení

Spracovávané stránky sa podľa mediánu presnosti delia do 5 kvalitatívnych skupín.Graf č. 2 zobrazuje kvalitatívne skupiny a predpokladanú presnosť OCR v každej z nich. Skupiny nie sú pevne stanovené. Rozsah skupín záleží na použitom OCR. Aby sme vedeli stránku zaradiť do správnej skupiny musíme stránku spracovať rôznymi systémami a pre každý zistiť presnosť. Zo získaných dát vypočítame médian, podľa ktorého stránke priradíme skupinu. [6]



Obr. 2: Závislosť presnosti na kvalitatívnej skupine pre Recognita OCR

5.3 Tesseract

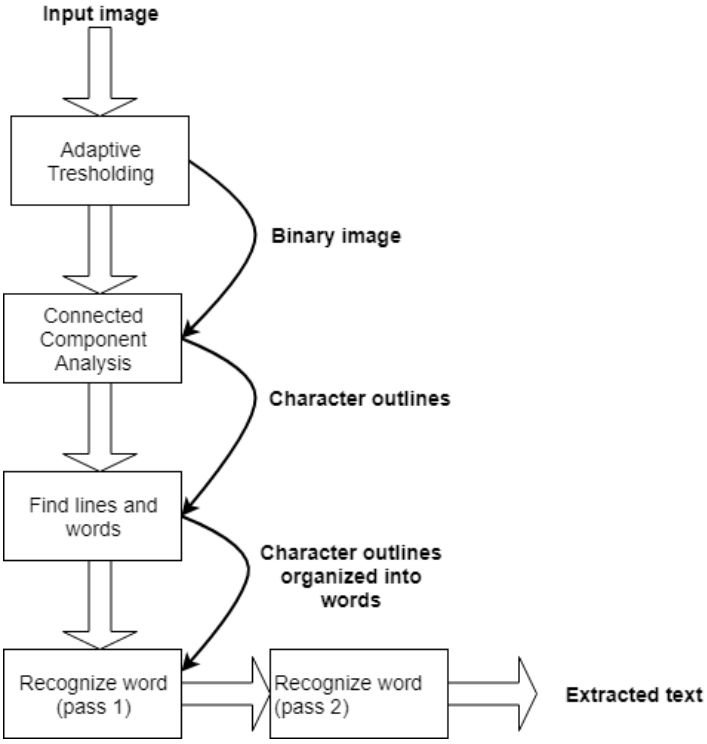
Open source knižnica na rozpoznávanie textu, ktorá bola vyvinutá firmou HP v rokoch 1984 - 1994. V roku 1995 sa Tesseract prvý krát predstavil na každoročnom teste OCR systémov, ktoré sa konali na univerzite v Nevade pod záštitou ISRI. Tesseract začal ako Phd. projekt v laboratóriach HP. Motiváciou bola skutočnosť, že vtedajšie OCR neboli ešte tak dobre optimalizované a zlyhali takmer na všetkom čo malo horšiu kvalitu. Projekt mal významný náskok pred komerčnými OCR. HP Labs držali tento projekt v utajení a nikdy sa nestal ich produktom. Druhá fáza vývoja bola viac zameraná na efekt odmietnutia[5] než na presnosť. Vývoj skončil na konci roku 1994. V roku 1995 bol projekt odoslaný na každoročný test. Projekt zožal úspech a dokázal, že sa môže rovnať komerčným OCR tej doby. Až v roku 2005 HP vydalo OpenSource verziu Tesseractu, ktorá je dostupná pod Apache 2.0 licenciou na adrese [6]. Tesseract môže byť použitý priamo v kóde alebo ako API a je nezávislý na platforme. Grafické rozhranie musí riešiť samotný programátor podľa spôsobu použitia programu, pretože Tesseract nemá vstavané grafické rozhranie. [4] [5]

5.3.1 Architektúra Tesseractu

Tesseract pracuje krok za krokom podľa blokového diagramu 3 podľa [3]. Prvým krokom je Tresholding, ktorý konvertuje obrázok do binárnej podoby. Ďalším krokom je analýza pripojených komponentov, ktorá sa používa na extrakciu znakov. Metóda je veľmi užitočná, pretože spracováva biely text na čiernom pozadí. Tesseract bol pravdepodobne prvý systém, ktorý ponúkol takýto druh spracovania. V ďalšom kroku sa obrysy znakov skladajú do slov a riadkov. Po zložení sa spustí rozpoznávanie podľa princípu dvoch prechodov, ktoré je popísané vyššie v kapitole 5.2. Detailnejšie informácie o každej fáze sú dostupné na [4]

[5] Väčšinou realizovaný ako neurónova sieť, ktorá na základe naučených dát a nejakého vstupu vráti najpravdepodobnejší výstup alebo ho odmietne. Sieť môže byť naučená na krstné mená a rozhoduje či vstup je krstné meno alebo nie. [2]

[6] `http://code.google.com/p/tesseractocr`



Obr. 3: Architektúra Tesseract

6 Použité knižnice

Všetky použité knižnice v projekte sú OpenSource. Ide hlavne o knižnice na prácu s grafikou pre spracovanie a úpravu obrázkov. Na to som použil knižnice OpenCV a Leptonica. Ďalšou zaujímavou knižnicou je GhostScript, ktorý slúži na prevod súborov typu pdf na png.

6.1 Leptonica

Leptonica je OpenSource knižnica obsahujúca softvér, ktorý je všeobecne používaný pre aplikácie spracovania obrazu a analýzy obrazu. Oficiálne repository pre leptonicu je na stránke[7]. Viac informácii o tejto knižnici nájdete na stránke[8].
Leptonica ponúka niekoľko základných operácii s obrázkami:

- Raterop
- Transformácie (škálovanie, rotácia, strihanie) na obrázkoch ľubovoľnej hĺbky pixelov
- Bilineárne transformácie
- Binárna a čierno-biele transformácie
- Transformácie obrázkov zo zmenou hĺbky pixelov v rovnakej alebo zmenenej mierke
- Pixelwise maskovanie, miešanie, vylepšenie, aritmetické operácie.

OpenSource projekty využívajúce Leptonicu:

- Tesseract
- OpenCV
- jbig2enc

6.2 OpenCV (OpenSource Computer Vision Librarz)

OpenCV bol navrhnutý ako cross-platform knižnica. Je napísaný v jazyku C, čo zabezpečuje prenositeľnosť na takmer všetky OS. Knižnica je dostupná pod licenciou BSD a preto je bezplatná pre akademické aj komerčné využitie. Podporovaná je väčšina OS (Windows, Linux, Mac OS, iOS, Android). Bol navrhnutý pre výpočtovú efektívnosť a zo zameraním na aplikácie v reálnom čase. Vďaka tomu, že knižnica je napísaná v optimalizovanom C/C++ môže využiť viacjadrové spracovanie. V kombinácii s OpenCL(framework pre písanie programov, ktoré sa budú vykonávať na heterogénnych platformách) môže využiť hardvérovú akceleráciu základnej heterogénnej počítačovej platformy. [1]

[7] `https://github.com/DanBloomberg/leptonica`
[8] `http://www.leptonica.com/`

6.3 GhostScript

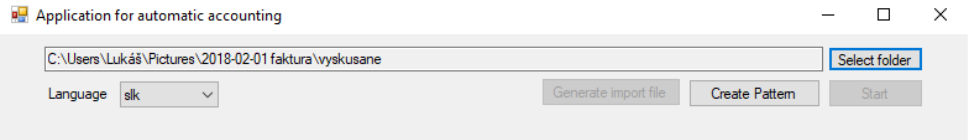
GhostScript je interpreter pre súbory PostScript a PDF. Je k dispozíci pod GNU GPL licenciou, ale aj ako komerčný produkt firmy Artifex. Je vyvíjaný počas posledných 20 rokov a za ten čas bol prenesený do rôznych systémov. Skladá sa z PostScriptovej interpreter vrstvy a grafickej knižnice. Niekedy je grafická knižnica nesprávne označovaná ako samotný GhostScript. Binárne súbory pre GhostScript pre rôzne systémy je možné stiahnuť z webovej adresy[9]. Zdrojový kód je súčasťou sťahovania.[10] Popis interpreterov, ktoré knižnica obsahuje:

- GhostPDF je interpreter na vrchole GhostScript, ktorý spracováva súbory vo formáte PDF. Spolieha sa na rozšírenie postScriptového jazykového/zobrazovacieho modelu, a preto ho nemožno použiť nezávisle od GhostScript.
- GhostPLC interpreter pre PLC a PXL súbory. Skladá sa z PCL/PXL prekladača pripojeného ku grafickej knižnici GhostScript
- GhostXPS je interpreter pre súbory XPS. Skladá sa z XPS prekladača pripojeného ku grafickej knižnici GhostScript
- GhostPDL (Ghost Page Description Languages) je zastrešujúcim termínom pre vyššie uvedené technológie. Všetky uvedené prekladače sú postavené na rovnakej grafickej knižnici.

[9] `www.ghostscript.com/download`
[10] `https://www.ghostscript.com/`

7 Zadaná úloha a riešenie

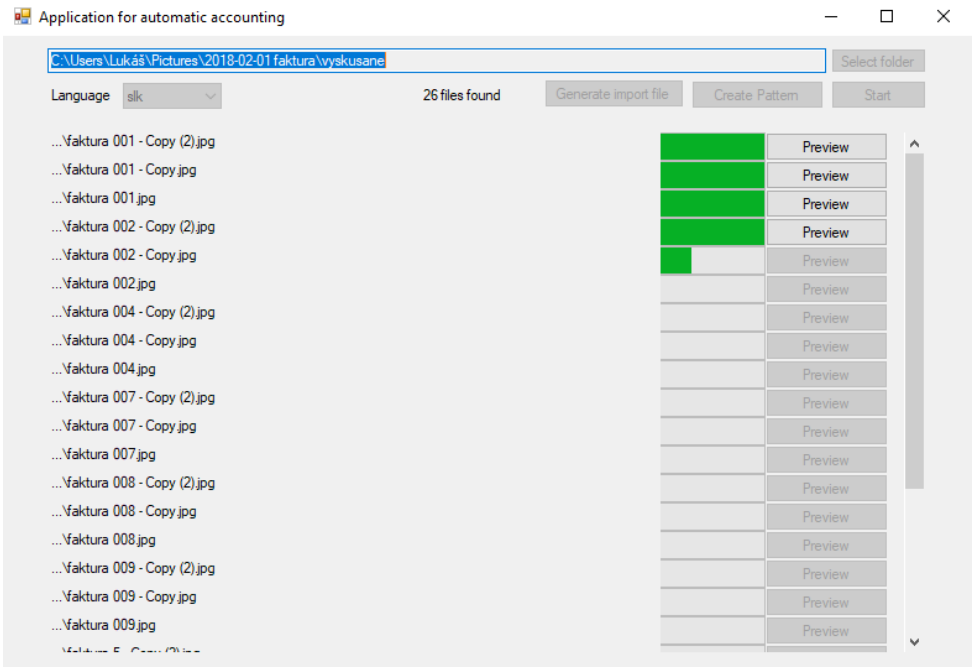
Mojou úlohou bolo vytvoriť desktopovú aplikáciu, ktorá bude načítať, spracovávať a po spracovaní daný doklad importovať do účtovníckeho softvéru. Program je zameraný na doklady fakturácie. Aplikacia je schopná spracovávať rôzne typy obrázkov ako napríklad bmp, jpg, jpeg, png, gif, tiff a taktiež súbory formátu pdf. Po spracovaní má užívateľ možnosť skontrolovať rozpoznané dáta a uložiť ich do databázy ako vzor pre ďalšie spracovávanie. Pred samotným spracovaním súboru sa program pozrie do databazy a podľa pozícii skontroluje, či má takýto vzor uložený. Podľa toho sa rozhodne, či bude súbor spracovávať podľa tohto vzoru alebo štýlom "BruteForce". Databáza nie je nutnosť pre aplikaciu slúži len pre ukladanie vzorov. Ak nie je pripojená program vždy pôjde štýlom "BruteForce".



Obr. 4: Užívateľské rozhranie po spustení programu

Po zobrazení základného UI obr. 4 má užívateľ dve možnosti. Môže si vytvoriť vlastný vzor pre nejakú faktúru(o tejto možnosti viac v kapitole 8), ktorá sa mu často opakuje alebo vybrať zložku s faktúrami a spustiť spracovávanie. Pred samotným spracovaním je nutné vhodne zvoliť jazyk, v ktorom bude text rozpoznávaný. Jazyk nie je možné dodatočne zmeniť. V programe sú dostupné jazyky Slovenčina, Čeština, Angličtina. Spracovať doklad v inom jazyku nie je problém, stačí si z adresy[11] stiahnuť súbor pre daný jazyk a vložiť ho do zložky tessdata pri projekte. Po zvolení zložky s obrázkami sa UI rozšíri o zoznam nájdených obrázkov. Každý súbor má priradený progress bar, ktorý zobrazuje priebeh spracovávania a tlačidlo pre zobrazenie náhľadu obrázka a údajov, ktoré sa podarilo rozpoznať. Na obrázku 5 môžeme vidieť program spracovávať dokumenty.

[11] `https://github.com/tesseract-ocr/tessdata/tree/3.04.00`



Obr. 5: Užívateľské rozhranie po spustení programu

Na obrázku 5 môžeme vidieť, že súbory sa spracovávajú postupne ako sú zoradené v zozname. Vnútorne sú však súbory spracovávané asynchrónne teda neblokujú hlavné vlákno a počas spracovávania vybraných dokumentov, možme bez ovplyvnenia rozpoznávania vytvárať vzory alebo kontrolovať už spracované súbory. Po spracovaní všetkých dokumentov sa sprístupní tlačidlo, ktorým sa spustí proces generovania importného súboru. Tento súbor je uložený pri projekte v zložke Ëxportäko ïnvoice_DD/MM/YYYY.txt". Súbor má špecifické usporiadanie hodnôt pre ekonomický softvér Omega.

Niektoré obrázky môžu mať zlé rozlíšenie, kontrast, môžu obsahovať šum atď. Všetky nežiadúce efekty musíme odstrániť pred spustením rozpoznávania aby bol výsledok čo najlepší. Na toto som použil knižnicu OpenCV, ktorá je zameraná práve na prácu s grafikou.

Použil som niekoľko funkcii:

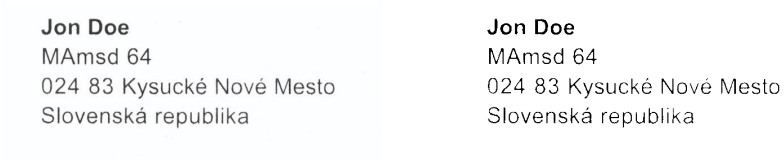
- Cv2.Treshold - funkcia zvýrazní text. Výsledok môžeme vidieť na obrázku 6
- DeskewImage - funkcia[12], ktorá otočí obrázok podľa zakrvenia textu. Implementované pre prípad ak je doklad zle naskenovaný alebo odfotený. Výsledok môžeme vidieť na obrázku 7
- RotateImageByTextOrientation - funkcia, ktorá otočí obrázok podľa toho ako je orientovaný text. Orientáciu textu nie je možné zistiť z vlastností obrázka, a preto som implemen-

[12] `http://mdb-blog.blogspot.cz/2010/10/c-how-to-deskew-image.html`

toval funkciu 1, ktorá v prvom kroku zistí o koľko stupňov musíme obrázok otočiť, aby bol text správne orientovaný a v druhom kroku obrázok otočí. Funkcia spustí 4 vlákna, každé dostane ako parameter uhol, o koľko má obrázok otočiť a pomocou Tesseract-u zistí úspešnosť rozpoznania. Vlákno, ktoré bude mať najväčšiu úspešnosť otočilo obrázok o správny počet stupňov. Tento uhol je uložený v triede Orientation a po ukončení všetkých vlákien pomocou cyklu zistím, o ktoré vlákno sa jedná, a aký uhol bol použitý. Nakoniec ešte obrázok otočím. Výsledok funkcie si môžme pozrieť na obrázku 8.

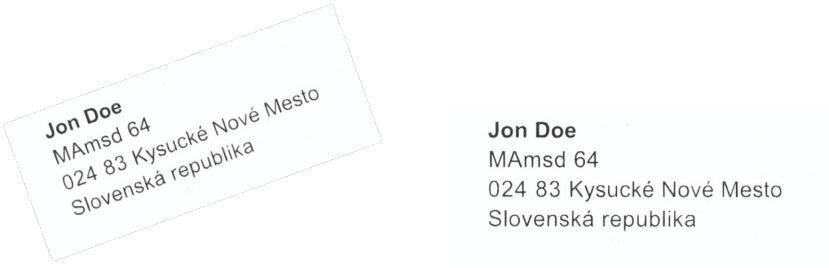
```
private void RotateImageByTextOrientation(ref Mat img, string lang)
{
   Orientation[] myThread = new Orientation[4];
    for (int i = 0; i < myThread.Length; i++)
   {
      myThread[i] = new Orientation(i * 90, img.Clone(), lang);
   }

   while (!myThread[0].Finished || !myThread[1].Finished || !myThread[2].
       Finished || !myThread[3].Finished)
      continue;

   Orientation max = myThread[0];
   for (int i = 1; i < myThread.Length; i++)
    {
      if (myThread[i].Confidence > max.Confidence)
        max = myThread[i];
   }
   RotateImage(img, ref img, max.Angle, 1);
}
```

Výpis 1: Metóda otočí obrázok podľa orientácie textu



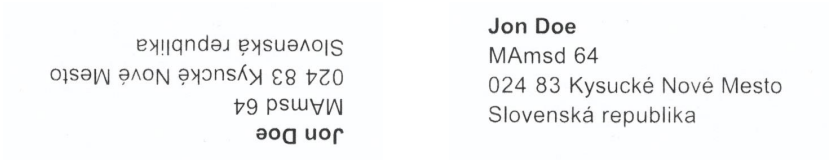
(a) Originál (b) Po spracovaní

Obr. 6: Ukážka funkcie Cv2.Treshold



(a) Originál (b) Po spracovaní

Obr. 7: Ukážka funkcie DeskewImage



(a) Originál (b) Po spracovaní

Obr. 8: Ukážka funkcie RotateImageByTextOrientation

Výsledkom procesu je obrázok v lepšej kvalite, odstránený sklon textu a otočenie obrázka tak, aby bol text orientovaný zľava do prava.

Program dokáže spracovať aj súbory typu pdf. V takom prípade sa žiadna z vyššie uvedených funkcii nevykonáva, pretože nie je potreba. Samotné pdf je v dobrej kvalite a aj správne orientované. Ak program narazí na takýto typ súboru, vykoná sa funkcia na prevod pdf na png a pokračuje ďalej. Na výstupe 2 je zobrazená funkcia, ktorá pracuje s pdf súbormy.

```
 try
{
   int desired_x_dpi = 200;
    int desired_y_dpi = 200;
   _rasterizer = new GhostscriptRasterizer();
    _rasterizer.Open(filePath, gvi, true);
   for (int pageNumber = 1; pageNumber <= _rasterizer.PageCount; pageNumber++)
   {
      Image img = _rasterizer.GetPage(desired_x_dpi, desired_y_dpi, pageNumber)
          ;
       images.Add(img);
   }
    _rasterizer.Close();
```

```
    MergeImages(images, finalPath);
}
catch (Exception ex)
{
    finalPath = string.Empty;
     if (_rasterizer != null)
       _rasterizer.Close();
    throw new Exception("Failed to initialize GhostScript!", ex.InnerException);

}
```

Výpis 2: Metóda na prevod .pdf na .png

Na začiatku si definujeme dpi. Pre našu potrebu stačí 200dpi. Cyklom prechádzame cez jednotlivé stránky pdf a pomocou knižnice ich konvertujeme na obrázky, ktoré ukladáme do listu. Jedna faktúra nemože byť na viacerých obrázkoch, pretože pri spracovaní nevieme ako bol obrázok získaný a či je doplnkom k inému alebo nie. Súvisiace obrázky by sme museli tagovať a ukladať tak, aby sme ich pri spracovávani vedeli identifikovať. Ja som zvolil menej náročnú cestu a obrázky z listu som spojil do jedného tak, aby boli pod sebou pomocou funkcie MergeImages.

Po kompletnom a úspešnom predspracovaní obrázka môžme pristúpiť k ďalšiemu kroku a tým je OCR. V našom prípade je to Tesseract OCR. Tento engine som vybral, pretože je dostupný pod Apache 2.0 licenciou, má dobrú úspešnosť v rozpoznávaní a podporuje viac ako 60 jazykov. Na výstupe 3 je zobrazená práca s enginom.

```
private int RunTesseract(Mat img)
{
    int conf = 0;
     using (TessBaseAPI engine = new TessBaseAPI(@".\tessdata", _lang,
         OcrEngineMode.TESSERACT_CUBE_COMBINED))
    {
        engine.InitForAnalysePage();
        engine.Init(null, _lang);
        engine.SetImage(new UIntPtr(BitConverter.ToUInt64(BitConverter.GetBytes(
            img.Data.ToInt64()), 0)), img.Size().Width, img.Size().Height, img.
            Channels(), (int)img.Step1());

        engine.Recognize();
        ResultIterator iterator = engine.GetIterator();
        IterateFullPage(iterator, ref _textLines);
        iterator.Dispose();
```

```
      conf = engine.MeanTextConf;
    }
  return conf;
}
```

Výpis 3: Metóda spustí rozpoznávanie pre daný obrázok

Funkcia na výstupe 3 inicializuje OCR. V zložke /tessdata sú uložené súbory dôležité pre OCR, kde je definovaná abeceda, písmená v rôznych fontoch, číslice, slová a pod. Tieto súbory je možné voľne upravovať alebo definovať, ako sa dané znaky majú preložiť. Súbory majú formát slk.traineddata. Premenná _lang v sebe uchováva trojmiestny názov jazyka, v ktorom sa bude rozpoznávať. Podľa tohto názvu sa vyberie správny súbor s abecedou. Tesseract definuje šesť módov rozpoznávania[13]. Ja som zvolil TESSERACT_CUBE_COMBINED, pretože má najvyššiu presnosť. Metóda IterateFullPage prechádza stránku po riadkoch. Pre každý riadok zistí text, ktorý sa v ňom nachádza, súradnice a veľkosť daného riadku. Rovnaké informácie sa ukladajú o slovách v riadkoch. Na konci je potrebné sa postarať o uvolnenie pamäte.

Ďalším krokom programu je spracovanie výstupu OCR. Existuje veľké množstvo vzorov a štýlov dokladov, ktoré bolo treba ošetriť. Počas práce na projekte som navrhol a implemenetoval niekoľko spôsobov vyhľadávania v texte. Každý som skúšal na testovacej množine dokladov a vybral najlepší:

- Definicia pozíc
  Každý doklad má rôzne rozmiestnenie kľúčových slov aj hodnôt, a preto definovať pozície a veľkosti jednotlivých slov pre každý doklad nie je ten najlepší prístup. Napriek svojej náročnosti bol algoritmus implementovaný kvôli špeciálnym typom dokladov a je bližšie popísaný v kapitole 8.2.

- Dvojbodka za kľúčom
  Väčšina párov (kľúč, hodnota) sú oddelené dvojbodkou. Algoritmus rozdelil riadok do poľa podľa znaku ':'. Prvá hodnota sa nachádza na pozícii 1(indexujeme od nuly) v poli. Táto pozícia obsahuje okrem hodnoty aj kľúč k ďalšej hodnote, ktorý musíme oddeliť. Ak deliaci znak nebol dobre rozpoznaný, tak môže obsahovať aj niekoľko kľúčov a hodnôt. V určitých prípadoch algoritmus bude fungovať. Sú to prípady, keď všetky kľúče majú hodnoty a sú rozdelené deliacim znakom, a zároveň znak je správne rozpoznaný. Tento spôsob je lepší nie však dostačujúci. Zlyhal takmer vždy, keď bol znak zle rozpoznaný alebo kľúč nemal hodnotu. OCR môže dvojbodku rozpoznať ako bodko - čiarku alebo iný znak a tým by bolo hľadanie údajov pre dané kľúčové slovo neúspešné.

- Prvé písmeno kľúča
  Algoritmus prechádzal slovník kľúčov, a pre každý z nich vyhľadal prvý výskyt prvého

[13] https://github.com/tvn-cosine/tesseract.net/blob/master/tesseract.net/Enums/OcrEngineMode.cs

písmena z kľúča. Potom od nájdenej pozície zobral podreťazec s dĺžkou kľúča. Tento podreťazec porovnal s kľúčom a ak sa zhodovali na viac ako 70% považuje ho za kľúč. Program pokračuje hľadaním hodnoty v texte napravo od nájdeného kľúča. Text môže obsahovať ďalšie kľúče. Pre istotu zavolám algoritmus ešte raz. Výsledkom je, že riadky sú prehľadávané zľava do prava, ale hodnoty sa ukladajú sprava do ľava. Ak nenájdem žiaden kľúč, tak uložím hodnotu a vymažem uložený text z riadku, aby sa neuložil v predchádzajúcom volaní algoritmu. Tento spôsob zlyhal v prípadoch keď slovník kľúčových slov obsahoval viac slov s rovnakým začiatočným písmenom. Prípady keď riadok obsahoval písmeno viac krát bol tiež problém.

- Prvé písmeno kľúča (všetky výskyty)
  Ako finálne riešenie tohto problému by som označil prístup algoritmu popisovaného v predošlom bode s úpravou vyhľadávania v riadkoch. Úprava spočíva v tom, že algoritmus vyhľadania kľúča sa vykoná pre všetky výskyty prvého znaku z kľúča. Pre každý výskyt a nájdený reťazec zistím podobnosť tohto textu s kľúčovým slovom zo slovníka. Po nájdení zhody pokračujem v podobnom duchu ako v predošlom bode. Tento algoritmus je popísaný na výstupe 6

Hlavička dokladov obsahuje údaje o odberateľovi, dodávateľovi, ktorí nemajú údaje v riadkoch, ale pod sebou. Pre takéto prípady musíme definovať, že nasledujúce riadky sa budú prehľadávať a ukladať inak.

```
private bool GoLikeColumn()
{
   var col = _listOfColumns.Where(c => c.Completed == false).FirstOrDefault();
   return col == null ? false : true;
}
```

Výpis 4: Metóda zisťuje či je nejaký stĺpec aktívny

Pre tieto prípady musíme definovať šírku stĺpca, v ktorom sa údaje nachádzajú podľa už existujúcich stĺpov alebo textu v riadku. Výšku nie je možné určiť dopredu. Musíme pokračovať, pokiaľ nenarazíme na hodnoty, ktoré už nepatria do stĺpca. Potom stĺpec ukončíme. Algoritmy sú dosť podobné. Rozdiel je v tom, že prvé riadky v stĺpcoch obsahujú údaje ako meno, ulica, štát. Tieto údaje netvoria pár (kľúč, hodnota) kvôli čomu nemôžeme použiť spomínaný algoritmus, ale musíme ich spracovať samostatne. Tieto údaje majú takmer vždy rovnaké usporiadanie, kde prvý riadkok obsahuje meno, druhý ulicu, tretí psč a mesto a posledný štát. Toto usporiadanie je v programe fixne definované. Ak sú údaje štruktúrované inak musíme si vytvoriť vzor.
Ak je podobnosť väčšia ako 70% program pokračuje odstránením špeciálnych znakov funkciou Trim(). Zvyšok riadku tvorí hodnotu pre daný kľúč. V tomto texte sa ešte môže vyskytovať iné kľúčové slovo, a preto sa algoritmus rekurzívne spustí znova. Algoritmus sa spúšťa dovtedy

kým sa v texte vyskytujú kľúčové slová. Niektoré doklady majú údaje (dátumy, konštantné, variabilné špecifické symboly) ukladané pod sebou. V prípade týchto kľúčov ak program nenájde hodnotu napravo od kľúča tak sa pokúsi nájsť hodnotu v stĺpci, ale maximálne o dva riadky nižšie. Dáta sa ukladajú pomocou reflexie. Ku každému kľúčovému slovu je priradená Property objektu, kde bude jej hodnota uložená. Pri nastavovaní hodnoty sa vykoná dodatočná validácia, ktorá je špecifická pre každú Property. Hodnoty sa zrkadlovo ukladajú do ďalšieho zoznamu. Tento zoznam obsahuje okrem kľúčov a hodnôt aj ich pozície a veľkosti. Zoznam sa používa pri náhľade a ukladá sa do databázy.

Na začiatku vyhľadávania je funkcia, ktorá prechádza všetky riadky dokladu. Funkcia kontroluje či sú nejaké stĺpce neukončené. Ak áno zavolá sa funkcia 5, ktorá prechádza všetky stĺpce. Funkcia vyberie z riadku text pre daný stĺpec podľa veľkosti stĺpca a súradnic slov v riadku. Môže sa stať, že stĺpec má nesprávne nastavenú ľavú alebo pravú x-ovú súradnicu a prechádza stredom slova. Takéto slová berieme ako keby patrili danému stĺpcu. Algoritmus sa spustí iba ak som v stĺpci našiel nejaký text. Po kontrole všetkých stĺpcov, ak zostal v riadku nejaký text spustím algoritmus znova zo slovníkom pre hlavičku. Je možné, že riadok už nepatrí stĺpcu a nachádzajú sa v ňom údaje hlavičky dokumentu. Ak v takomto texte nájdeme kľúč z hlavičky musíme ukončiť stĺpec nad ním, aby sa viac nekontroloval.

```
1.FillColumn(Line l, list of columns)
2. foreach column in columns
3. if (column.blocked)
4.    client = FindClient(column.id)
5.    textForColumn = FindTextForColuumn(column,l)
6.    if (textForColumn is empty)
7.    GetDataFromLine(l,textForColumn,dictionaryForClient,client,true,column)
8.       RemoveFromLine(l,textForColumn)
9. else
10.      next column
11.end of cycle
12.if (l.Text is not empty)
13.   GetDataFromLine(l,l.Text,generalDictionary,header,false,null)
```

Výpis 5: Metóda FillColumn popísaná pseudokódom

```
1.GetDataFromLine(Line r, Text text, Dictionary s,object objectForSave,
   TrueFalse processingColumn, object column, TrueFalse lookingForRightEdge)
2.foreach((key,value) pair in s)
3. firstOccurance = firstOccuranceOfCharInText(text.FirstOccurance(FirstChar(
   pair.key)))
```

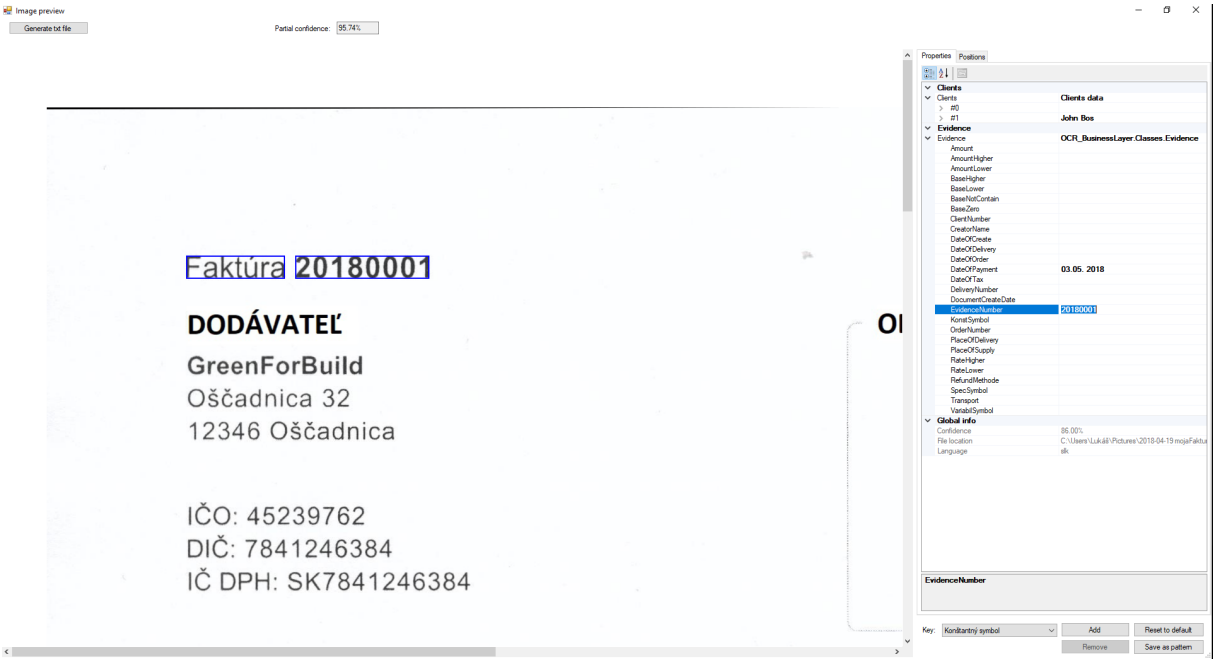
```
4. keyLength = pair.key.length
5. while(( keyLength + firstOccurance ) <= text.length a firstOccurance not
   equal -1)
6.    keyInText = text.GetText(firstOccurance ,keyLength )
7.    similarity = GetSimilarity(pair.key, keyInText )
8.    if (similarity > 70)
9.       if (column not empty and objectForSave typeof header)
10.           column.Finished = true
11.       SaveValue(r,pair,keyInText,text,firstOccurance,s,objectForSave,
   lookingForRightEdge,processingColumn)
12.      else
13.       selectedText = text.GetText(text.FirstOccurance(keyInText) + 1)
14.       index = selectedText.firstOccurance(pair.key)
15.       if (index == 0 and indexIsEmpty or index == -1)
16.          and of cycle
17.       if (index == 0)
18.          indexIsEmpty = true
19.       firstOccurance += index +1
20.   end of cycle
21.end of cycle
```

Výpis 6: Implementovaný algoritmus vyhľadávania v texte

8 Náhľad obrázkov a vlastné vzory

8.1 Náhľad obrázkov

Po vyťažení údajov a po spracovaní výstupného textu má užívateľ možnosť si zobraziť náhľad obrázka, ktorý spracovával a k nemu údaje. Po kliknutí na tlačidlo Preview v hlavnom formulári 5 sa užívateľovi zobrazí náhľad obrázka 9. V pravom paneli sú zobrazené všetky dôležité údaje. Panel obsahuje aj údaje o súbore, cestu k nemu, jazyk, v ktorom bol rozpoznaný a úspešnosť OCR. Údaje sú zobrazené pomocou PropertyGridu pre jednoduchú kontrolu a prehľadnosť. Program ponúka možnosť skontrolovať umiestnenie údajov. Po dvojkliku na nejaký údaj v pravom panel sa v obrázku vykreslí rám, odkiaľ boli údaje rozpoznané a v TextBox-e nad obrázkom sa zobrazí úspešnosť OCR na slovách v rámčeku. Užívateľ má možnosť tento rámček ľubovoľne premiestňovať a meniť veľkosť. Panel s údajmi je možné upravovať. V prípade, že užívateľ nie je spokojný s textom, môže ho prepísať. Po kontrole stačí vygenerovať importný súbor alebo uložiť pozície ako vzor do databázy. Pri generovaní program vždy vytvorí nový textový súbor.

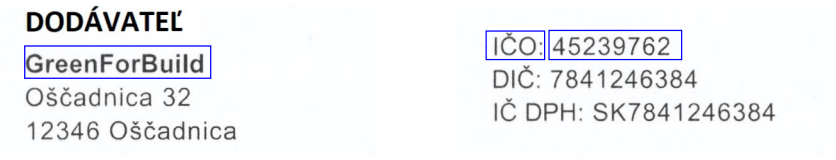


Obr. 9: Náhľad obrázka s vyťaženými údajmi

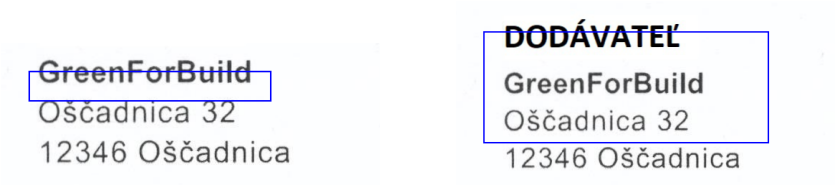
8.2 Vlastné vzory

Pre prípady dokladov, ktoré sa často opakujú som implementoval možnosť vytvoriť si na tieto dokumenty vzor. Náhľad UI pre vytváranie vzorov môžme vidieť na obrázku 12 Pri vytváraní je nutné mať pripojenú databázu, do ktorej sa budú ukladať pozície slov. Po vytvorení a uložení, sa bude kontrolovať každý ďalší dokument, či sa jedná o vzor alebo nie. Pri vytváraní vzorov

musíme dbať na správnosť kľúčov a k nim vybrať odpovedajúce hodnoty.



Obr. 10: Ukážka správneho definovania pozíc



Obr. 11: Ukážka nesprávneho definovania pozíc

Obrázky 10 a 11 zobrazujú, ako správne definovať pozície. Užívateľ vyberie z comboBoxu vpravo dole, aký kľúč ide vkladať a klikne na Add. Hodnoty (Meno, Adresa, Ulica, Štát) nemajú k sebe priradený kľúč. V takom prípade program dovolí zadať iba jedno pole. Po kliku na tlačidlo Add sa program prepne do režimu kreslenia a vyzve užívateľa, aby nakreslil požadovaný počet polí. Po dokreslení sa program automaticky prepne do režimu pred klikom. Pole kľúča by malo byť čo najmenšie a nemalo by obsahovať iný text ako kľúč. Môže obsahovať špeciálne znaky. Pole hodnoty by nemalo obsahovať text kľúča a kedže hodnota je premenlivá, šírka definovaného poľa môže byť väčšia. V pravom paneli je možné si definované polia preklikať a dodatočne upraviť alebo zmazať. Dostupná je aj funkcia generovania importného súboru. Po zadefinovaní polí kľúče ešte nemajú priradenú hodnotu. Po výbere tejto funkcie sa spustí rozpoznávanie a algoritmus hľadania hodnôt podľa práve vytvoreného vzoru. Výsledkom je importný súbor a užívateľovi sa zobrazí náhľad, ako je popísaný v kapitole 8.1. Dostupná je aj funkcia Save Pattern, ktorá definované pozície uloží do databázy. Opakované spracovanie tohoto dokumentu alebo podobného sa bude vykonávať podľa už vytvoreného vzoru.

Pri kontrole vzorov si z databázy zoberiem prvých 5 kľúčov z každého vzoru a skontrolujem, či pozície a text súhlasí s textom v obrázku. Musia to byť párové hodnoty (kľúč, hodnota). Pri zisťovaní, či dokument je vzor nemôžem kontrolovať hodnoty kľúčov, pretože tie sú premenlivé. Do úvahy beriem iba kľúče (IČO, DIČ, Variabilný symbol a pod.). Dané kľúče majú v databáze uložené pozície. V databáze je uložené aj rozlíšenie obrázka, z ktorého bol vzor vytvorený. Podľa neho musím prepočítať súradnice pretože môžem spracovávať rovnaký doklad v inom rozlíšení. Podľa nich vystrihnem časť z obrázka. Z tohto podobrázka musím rozpoznať text pomocou OCR a skontrolovať, či sa tam nachádza kľúč, ktorý by tam mal byť podľa databázy. V prípade, že program označí dokument ako dokument, pre ktorý platí daný vzor, nebude sa vykonávať zložitý

algoritmus vyhľadávania v texte ako bolo popísané v kapitole 7, ale pozrie sa len na miesta, ktoré má v databáze. Takýto spôsob získavania údajov je rýchlejší a presnejší. Pri zisťovaní hodnôt ku kľúčom je to o niečo zložitejšie. K hodnotám nemôžeme pristupovať spôsobom ako pri kľúčoch. Tesseract si vnútorne určuje výšku riadkov podľa veľkosti písma, fontu, medzier. Mohlo by sa stať, že vystrihnutý podobrázok rozpozná ako viac riadkov. V takom prípade nevieme, v ktorom riadku je relevantná hodnota ku kľúču a nemôžeme použiť spôsob hľadania kľúčov. Pre hodnoty som implementoval samostaný algoritmus. Pred hľadaním spustím rozpoznávanie nad celým obrázkom. Podľa vrchnej hrany obdĺžnika, v ktorom sa nachádza hodnota zoberiem riadok nad a pod hodnotou. Získal som dva riadky, v ktorých sa hodnota určite nachádza. Pre každý riadok vypočítam percentuálne prekrytie s obdĺžnikom hodnoty. Riadok s najväčším prekrytím má najväčšiu pravdepodobnosť, že sa v ňom nachádza hľadaná hodnota. Podľa X-ovej súradnice a šírky obdĺžnika hodnoty vyberiem z riadka text. Na výstupe 7 je zobrazená funkcia, ktorá vyberie text či už pre stĺpec alebo pre oblasť zadanú užívateľom.

```
public static string GetWordsForColumn(Column col, TextLine line)
{
   string colText = string.Empty;

   foreach (Word w in line.Words)
   {
      if (w.Bounds.Left < col.Right && w.Bounds.Width < CONSTANTS.
          MAX_LENGTH_OF_ONE_WORD)
      {
         if (((w.Bounds.Left <= col.Left && w.Bounds.Right > col.Left) || w.
             Bounds.Left >= col.Left) && ((w.Bounds.Right >= col.Right && w.
             Bounds.Left < col.Right) || w.Bounds.Right <= col.Right))

         {
            if (col.Left > w.Bounds.Left)
            col.Left = w.Bounds.Left;
            colText += w.Text + " ";
         }
      }
   }
return colText.Trim();

}
```

Výpis 7: Metóda vyberie slová z riadku medzi súradnicami Left a Right

Spomínaná funkcia rieši aj problém, ak by slovo presahovalo cez stĺpec a zoberie ho ako keby nepresahovalo za hranicu stĺpca. Z vybraného textu odstránim špeciálne znaky a uložím. Tým je spracovávanie ukončené a program sa posunie na ďalší doklad v poradí.

Obr. 12: Vytváranie vzorov

9 Záver

Odbornú prax hodnotím ako veľmi prínosnú a užitočnú. Mal som možnosť využiť znalosti získane počas štúdia, prevažne na predmetoch ktoré sa venovali vývoju a návrhu architektúri softvéru, ale aj predmety pojednávajúce o databázových systémch. Dozvedel som sa aj veľa nových vecí hlavne v oblasti práce s grafikou, OCR systémami. Všetky ciele práce sa mi podarilo splniť. Program dokáže spracovať text hlavičky dokumentov asi na 90% na testovacej skupine dokladov, ktorá obsahuje asi 20 dokladov. Iné typy dokladov nemusia dosahovať tak dobré výsledky, ale môžu dosahovať aj väčšie. Spracovávanie pomocou vzorov dosahuje úspešnosť do 98%. V prípade rozpoznávania údajov pre účtovníctvo si nemyslím, že je tento algoritmus vhodný. Taktiež spracovávanie pomocou šablón nevidím ako vhodné riešenie. Pri vývoji podobného programu by som sa zameral skôr na neurónové siete alebo machine-learning, ktorým patrí budúcnosť vývoja.

Literatúra

[1] Gary Bradski and Adrian Kaehler. Opencv. *Dr. Dobb's journal of software tools*, 3, 2000.

[2] Jon C Geist and R Allen Wilkinson. Ocr error rate versus rejection rate for isolated handprint characters. In *Character Recognition Technologies*, volume 1906, pages 267–279. International Society for Optics and Photonics, 1993.

[3] Chirag Patel, Atul Patel, and Dharmendra Patel. Optical character recognition by open-source ocr tool tesseract a case study. *International Journal of Computer Applications*, 55(10), 2012.

[4] Ray Smith. An overview of the tesseract ocr engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633. IEEE, 2007.

[5] Ray Smith. An overview of the tesseract ocr engine. `https://github.com/tesseract-ocr/docs/blob/master/tesseracticdar2007.pdf`, 2015.

[6] Frank R. Jenkins Stephen V. Rice and Thomas A. Nartker. The fourth annual test of ocr accuracy. `https://github.com/tesseract-ocr/docs/blob/master/AT-1995.pdf`, 2015.